COMMENT GAGNER SON PREMIER MILLIARD EN BUG BOUNTY

(FACILEMENT)

FÉLIX BILLIÈRES

LE RÉFLEXE

01 Je trouve un scope sympa sur YesWeHack

02 claude **--dangerously-skip-permissions**

03 "trouve-moi des bugs"

TRADUCTION...

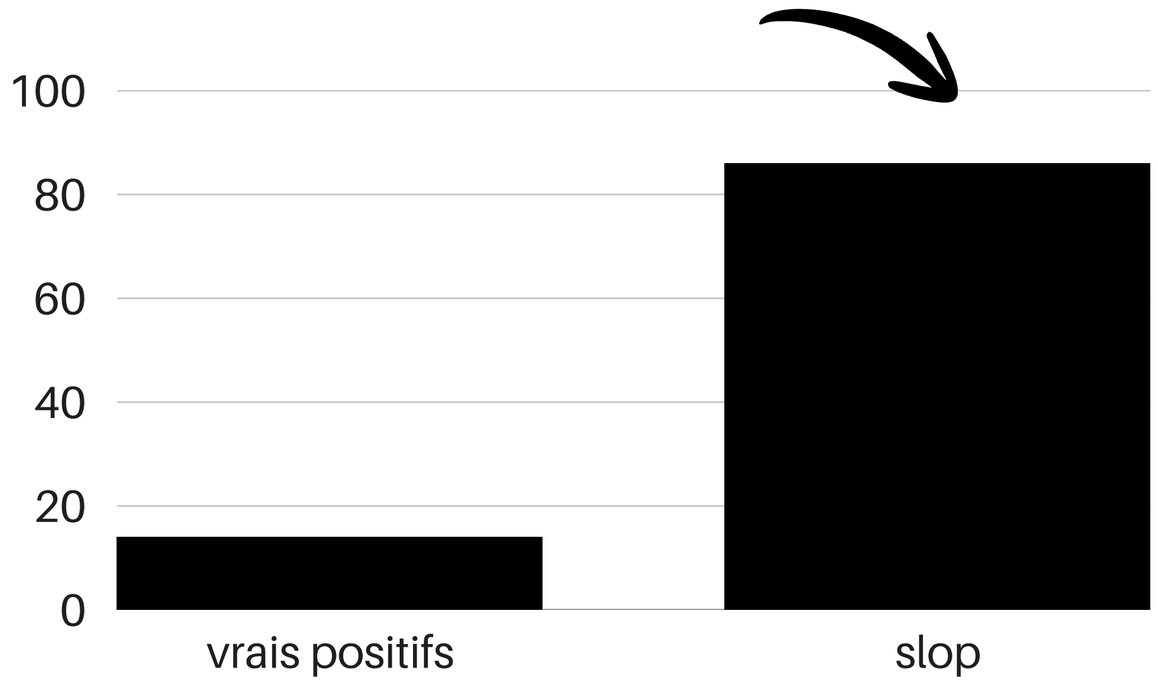
Are you sure?

Yes

Us

(yes, you AND me twin)

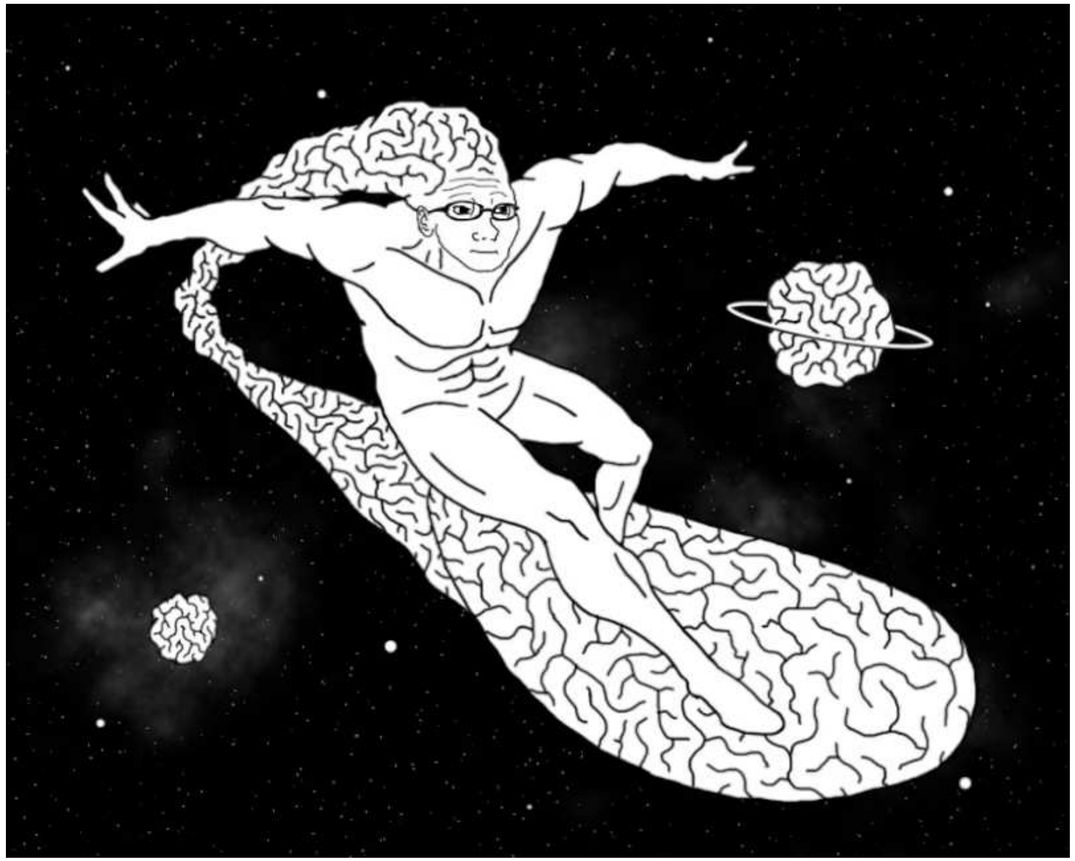
TRIAGER'S NIGHTMARE



- daniel.haxx.se/blog/2026/01/26/the-end-of-the-curl-bug-bounty/

RESULTAT

CONFIGURATION BEATS CAPABILITY



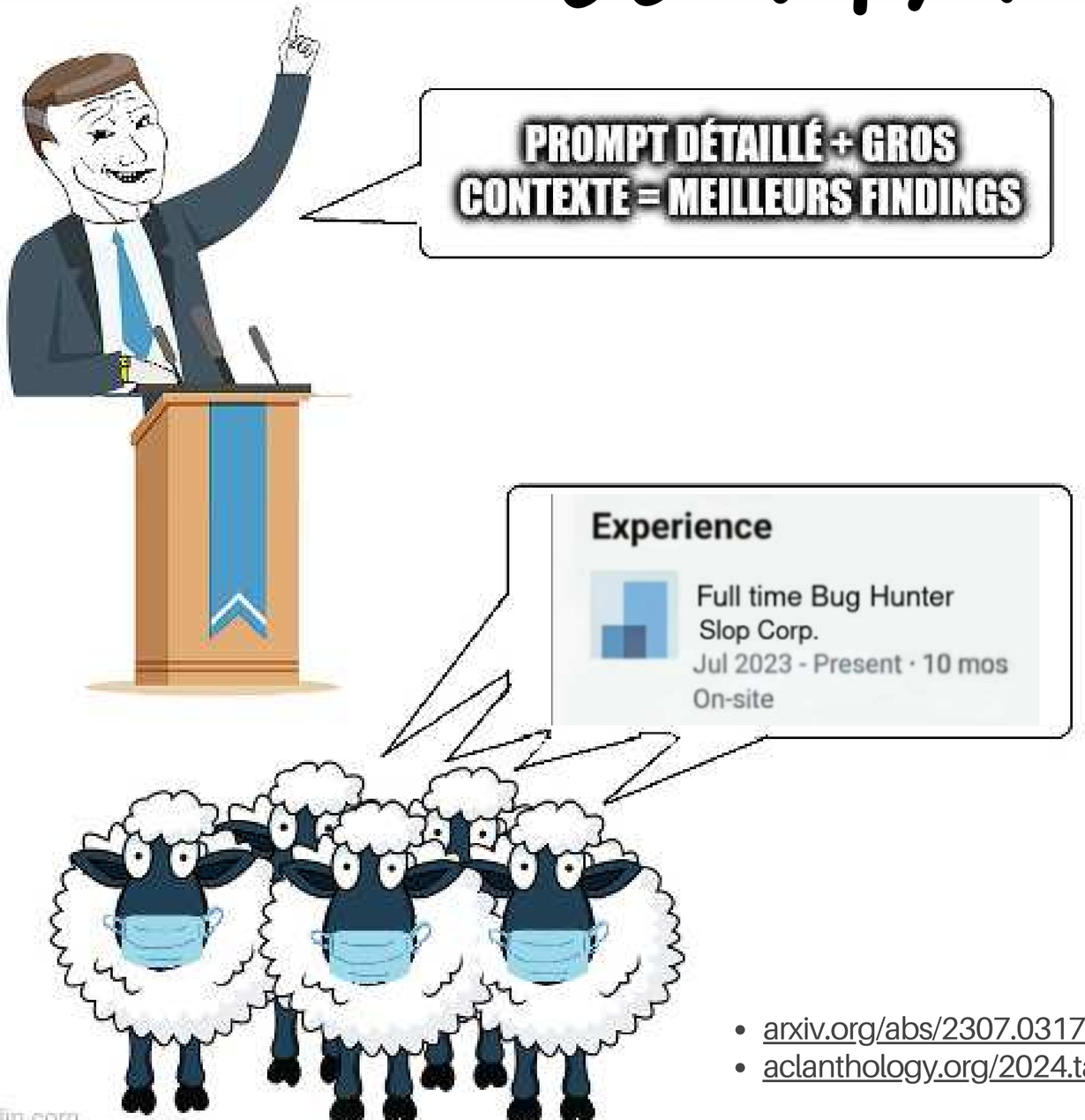
Le même modèle frontier (Claude / GPT-5.5 / Gemini) donne 14 % ou 80 %+ selon comment tu l'orchestres.

Le bug n'est pas dans le modèle. Il est dans le **harness**.

harness = toute la tuyauterie autour du modèle : prompts, outils, validateurs, orchestration.

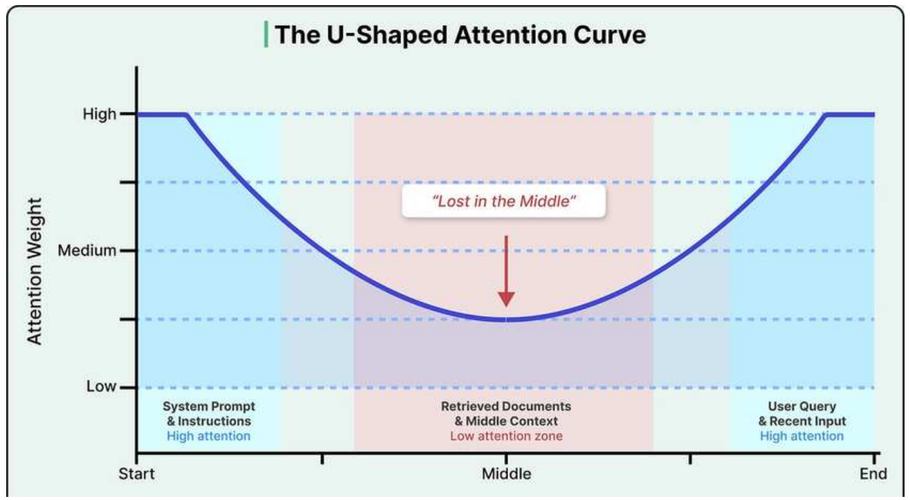
- strobes.co/blog/ai-harness-offensive-security-llm-pentest-architecture/
- xbow.com/blog/alloy-agents

LE MYTHE À TUER



Contexte	Trajectoires qui réussissent
27k tokens	**8%**
100k tokens	**1%**

8× pire en grossissant le prompt.



- arxiv.org/abs/2307.03172
- aclanthology.org/2024.tacl-1.9

2026 : LA CAPABILITY SE BANALISE

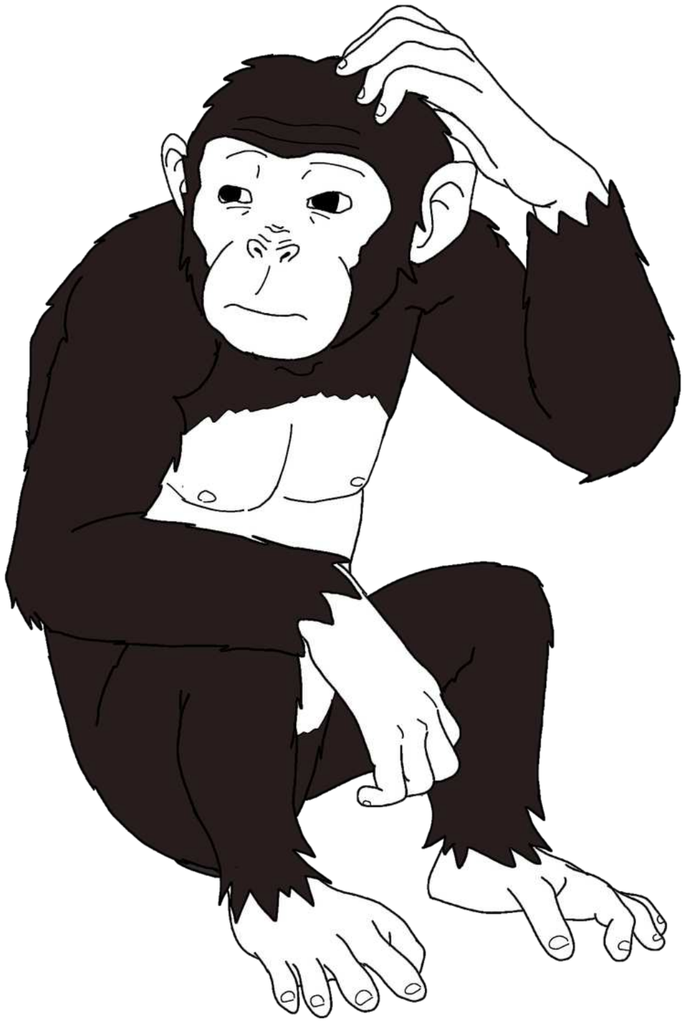
- ***Cybench :*** 17,5 % → 93 % en ~18 mois. medium/Hard CTF = minutes.

- ***AISLE :*** 12 / 12 zero-days OpenSSL (jan. 2026) dont un bug de 27 ans, raté par des décennies de fuzzing.

- arxiv.org/abs/2408.08926
- anthropic.com/research/building-ai-cyber-defenders
- aisle.com/blog/what-ai-security-research-looks-like-when-it-works
- schneier.com — AI found twelve new vulnerabilities in OpenSSL

Au même moment, curl ferme son programme (95 % de rapports invalides).

Même vague IA, deux issues opposées. La seule différence : le harness.



- daniel.haxx.se/blog/2026/01/26/the-end-of-the-curl-bug-bounty/

DÉCOUVERTE ≠ VALIDATION

« Creative AI discovers. Deterministic logic decides what is real. »
- **XBOW, Black Hat 2025**

Le LLM propose. Un oracle déterministe confirme.

Oracle = un test en code pur qui répond oui/non.
Zéro IA → zéro hallucination.

- xbow.com/blog/black-hat-2025

LA CONFIANCE N'EST PAS AUTO-DÉCLARÉE

Le score n'est pas « le LLM se note 0,9 ».
C'est une formule sur des preuves :

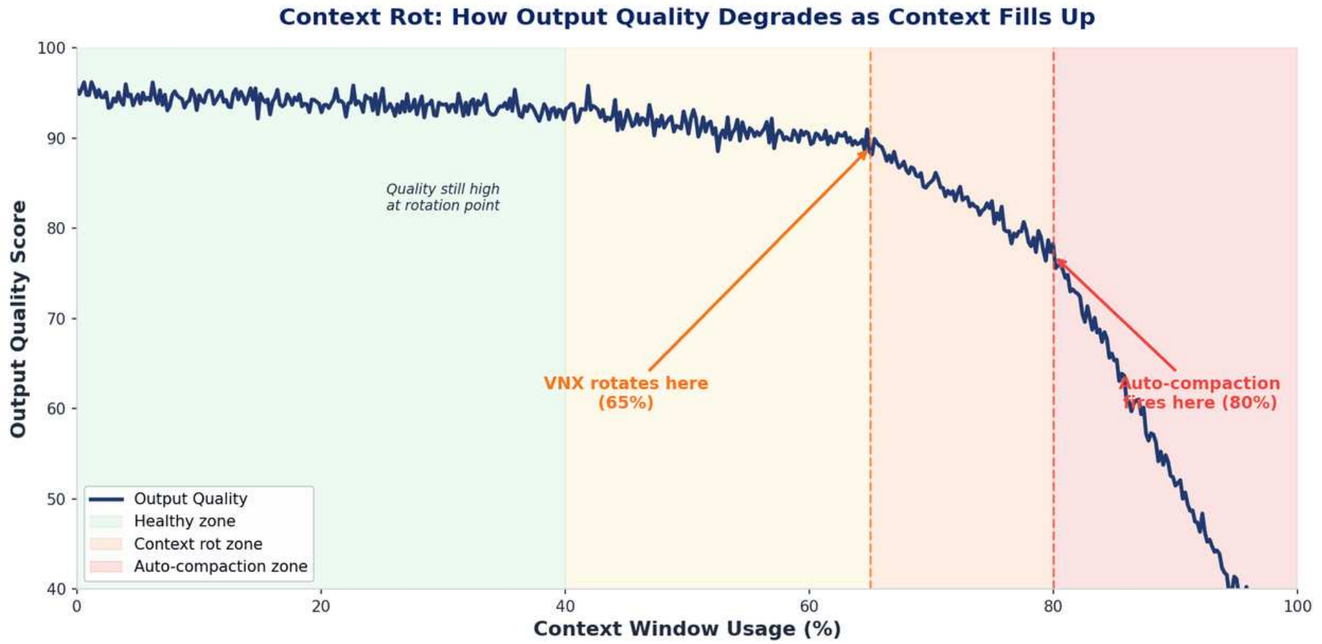
+0.30 oracle déterministe passé
+0.20 reproductible (min de la classe)
+0.15 contrôle négatif silencieux
+0.10 CVE/symbole vérifié (NVD + ripgrep)
-0.20 triche détectée
-0.50 check NVD/symbole échoué

---

seuil ≥ 0.85 → revue humaine

DISCIPLINE DE CONTEXTE

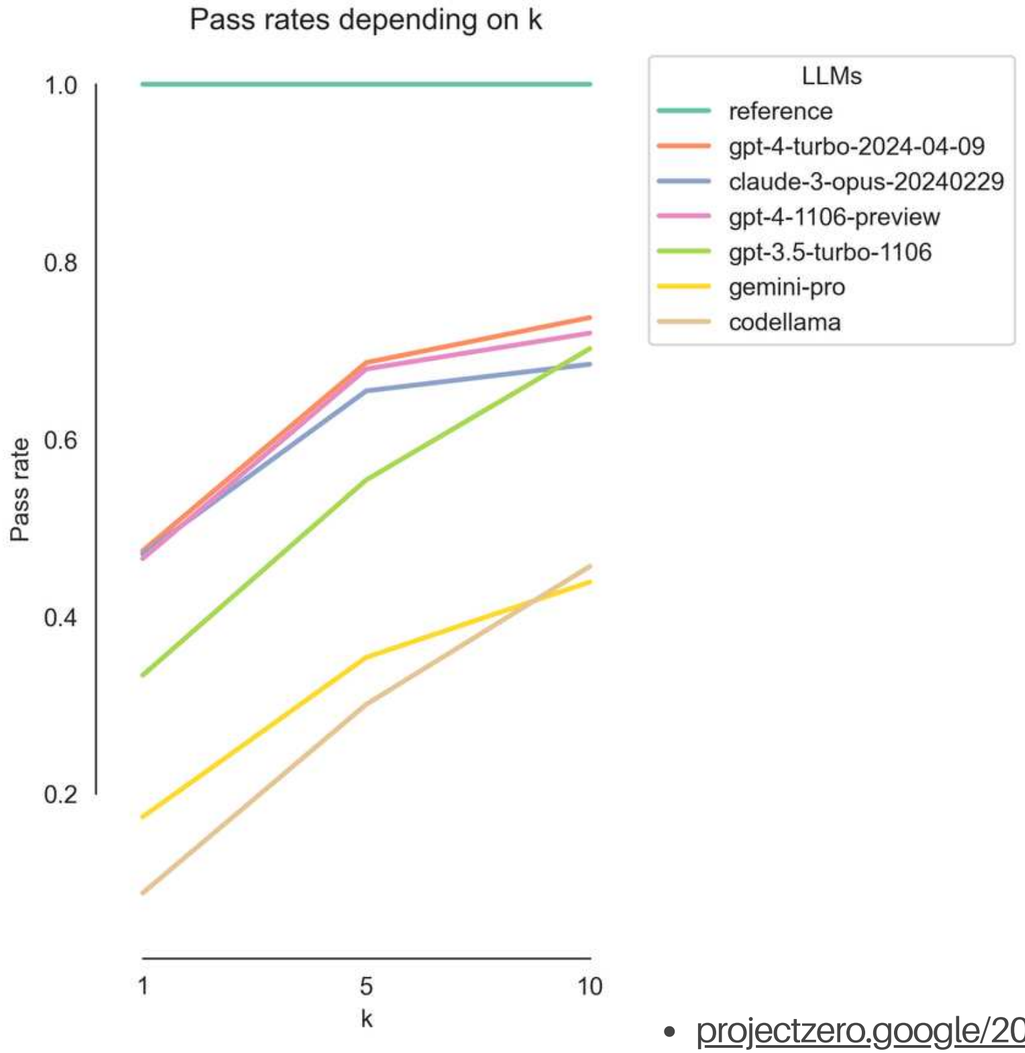
- 1 endpoint / 1 handler par tâche
- Sweet spot : 25–30k tokens par tâche d'audit
- Compacter à 60 % de capacité, pas à 95 %
- Clause magique : « Start every message w/ ‘Hello Master’ »



- trychroma.com/research/context-rot

PASS@K : 20 COURTS > 1 LONG

pass@k = proba de trouver le bug en k essais indépendants.



Même budget. Au lieu d'une longue trajectoire -> lance 20 trajectoires courtes et diverses.

Google Naptime, CyberSecEval2, GPT-4 Turbo :

0,05 → 1,00
pass@1 pass@20

TES SUBTASKS

- projectzero.google/2024/06/project-naptime.html

NIVEAU 2

ALLoY CRoSS-PRoVIDER

Alterner les modèles de providers différents dans la même boucle :

Setup	Performance
Sonnet 4.0 seul	57,50%
Sonnet 4.0 + Gemini 2.5 Pro alternés	68,80%

+11 points absolus. Alterner same-provider = 0 gain.
Il faut une vraie diversité d'erreurs. Gateway type LiteLLM.

- xbow.com/blog/alloy-agents

SPÉCIALISE PAR CLASSE, PAS PAR CIBLE

Un agent générique "trouve tous les bugs" = résultats moyens sur tout.

Agent spécialisé par classe avec son oracle dédié :

Architecture : **Coordinator** dispatche vers des specialists -> chaque specialist a sa propre mutation engine + oracle.

XSS = browser headless + nonce.
Blind SQLi = t-test de timing.
SSRF = callback OOB.
...

Classe	Générique (GPT-5)	Spécialisé (Sonnet 4)	Delta
XSS	57%	87%	+30 pts
Blind SQLi	33%	67%	+34 pts

- arxiv.org/abs/2508.20816

Ton agent ne sait pas qu'il galère

La vraie raison d'échec n'est pas les outils manquants.

Type A : outil absent, prompt mauvais → se corrige par ingénierie

Type B : l'agent sur-investit les branches faciles, épuise le contexte avant d'atteindre la chaîne intéressante → persiste même avec de meilleurs modèles

Solution : *Task Difficulty Assessment avant chaque sous-tâche.*

4 dimensions estimées en amont :

- **Horizon** -> combien d'étapes à parcourir ?
- **Evidence confidence** -> quelles preuves déjà disponibles ?
- **Context load** -> quel % du contexte déjà consommé ?
- **Historical success** -> taux de réussite sur cette classe ?

Score élevé → alloue un run pass@k dédié. Score faible → single-shot.

2/5 → 4/5 hôtes GOAD compromis

- https://arxiv.org/abs/2602.17622

LE LLM CHOISIT LES FEUILLES, PAS L'ARBRE

Comment forcer un LLM moyen à suivre une méthodo de senior :

Decision trees externalisés en YAML -> le LLM choisit seulement les feuilles, jamais la structure.

```
Endpoint analysé
│
├─ Accès / Auth
│    ├─ ID / référence ──────────── IDOR        → leaf: cross_session_diff
│    └─ Token / session ─────────── Auth bypass → leaf: cross_role_req
│
├─ Injection
│    ├─ HTML / JS ──────────────── XSS         → leaf: alert_nonce
│    ├─ Paramètre SQL ──────────── SQLi        → leaf: timing_welch
│    └─ Template engine ─────────── SSTI       → leaf: polyglot_probe
│
└─ Interaction réseau
     ├─ URL / path ──────────────── SSRF       → leaf: callback_oob
     └─ Redirect ────────────────── Open redir → leaf: redirect_check
```

Chaque feuille est un contrat YAML : oracle + params + budget.
Le LLM exécute, il ne conçoit pas.

```yaml
leaf.injection.timing_welch:        # SQLi blind
  oracle: welch_ttest
  params: { payload: "1' AND SLEEP(3)--", alpha: 0.01 }
  min_confidence: 0.85
  budget: { max_tool_calls: 8, timeout_s: 60 }

leaf.access.cross_session_diff:     # IDOR
  oracle: validate_idor
  params: { cross_session_check: true, require_distinct_auth: true }
  min_confidence: 0.85
  budget: { max_tool_calls: 6, timeout_s: 45 }
```

Llama-3-8B : 13,5 % → 71,8 % (+58 pts, modèle inchangé)

- strobes.co/blog/ai-harness-offensive-security-llm-pentest-architecture/

LE CIMETIÈRE

Ce qui NE marche PAS

(testé pour vous...)

R.I.P

Idée séduisante	Pourquoi c'est mort
Multi-agent debate naïf	Pire que single-agent à budget égal
RAG sur reports BB publics	90 % de slop : tu apprends à halluciner
27 skills "au cas où"	Picker burnout — cap à 8–12
10 sous-agents concurrents	Ban en minutes — cap à 4–6
Claude.md de 500 lignes	Adhérence chute après 200 lignes

- arxiv.org/abs/2604.02460
- arxiv.org/abs/2406.06461
- arxiv.org/abs/2605.24660
- tianpan.co/blog/2026-02-14-writing-effective-agent-instruction-files

RECAP DE LA RECETTE

1. Validator d'abord (oracle déterministe), sinon tu accélères du faux
2. Contexte serré, 25–30k, un handler par tâche
3. pass@k avec modèles pas chers, pas un long run Opus
4. Alloy cross-provider pour la diversité d'erreurs
5. Spécialise par classe, XSS agent ≠ SQLi agent ≠ SSRF agent
6. Task Difficulty Assessment, alloue le compute là où c'est dur
7. Un gate humain avant soumission